VISIT…

Несмотря на то, что есть еще в наше время люди, которые жестко различают понятия "хакер" и "взломщик", я часто употребляю эти понятия в книге в одном и том же контексте. Честно говоря, мне не очень интересны "религиозные войны", связанные с терминами, я просто решил в своей книге придерживаться официального мнения СМИ в данном вопросе. Хотя для меня лично хакер — это, прежде всего, тот, кто самостоятельно программирует, творит и созидает, а не ломает и использует чужие наработки в целях наживы и причинения ущерба кому-либо.

Все приводимые в книге примеры программ написаны для компьютера x86, работающего под операционной системой Linux. Там, где это было возможно, работоспособность программ была протестирована сразу в двух системах: Mandriva 2006 PowerPack (ядро 2.6.12) и Linux Red Hat 7.1 (ядро 2.4.2).

Вы можете изучать эту книгу как учебник последовательно, начиная с первой главы, или только выборочные главы в любом порядке.

Необходимые знания

Чтобы эта книга не показалась сухой и скучной, и чтобы вы могли получить подлинное удовольствие и пользу от ее прочтения, вам необходимо уже обладать некоторыми знаниями. Далее они перечислены в порядке повышения сложности с указанием источников, где их можно почерпнуть.

- Вы должны уметь пользоваться операционной системой Linux хотя бы на уровне простого пользователя. То есть вы должны уметь работать в консоли Linux и знать основные консольные команды, такие как: `ls`, `ps`, `who`, `man`, `cat`, `su`, `cp`, `rm`, `grep`, `kill` и т. д. Вы должны знать организацию файловой системы Linux и систему прав доступа; уметь регистрировать в системе новых пользователей и удалять старых; уметь пользоваться каким-нибудь текстовым редактором в Linux, например, vi[1]; уметь настраивать сеть и выход в Интернет. В общем, вы должны знать все, без чего невозможна полноценная работа в Linux. Для этого вам лучше приобрести книгу для начинающего пользователя Linux (подобных книг сейчас продается неисчислимое множество) и прочитать ее от корки до корки, параллельно экспериментируя на какой-нибудь установленной системе.
- Так как большинство приложений, о которых пойдет речь в этой книге, являются сетевыми, то вы должны четко знать базовые принципы построения локальных и глобальных компьютерных сетей. То есть необходимо иметь представление о различных топологиях сетей, уровнях модели OSI, стеке протоколов TCP/IP, адресации в IP-сетях, работе основных сетевых протоколов, стандарте Ethernet, принципах работы различных коммуникационных устройств, таких как концентратор, коммутатор, маршрутизатор и пр. Все эти базовые знания можно получить в следующей отличной книге: Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы. 3-е изд. — СПб.: Питер, 2005.
- Почти все исходные тексты программ для этой книги написаны на языке программирования Си, поэтому вы должны хорошо владеть этим языком. Фундаментальным учебником по Си является книга от самих создателей этого языка: Керниган Б., Ритчи Д. Язык программирования Си. 3-е изд., испр. — СПб.: Невский Диалект, 2001.
- Но хорошего знания языка Си еще не достаточно — вам необходимо уметь программировать под операционную систему Linux, т. е. вы должны знать особенности этой операционной системы с точки зрения программиста, должны знать и уметь применять стандартные библиотеки и функции Linux и т. д. Здесь я могу посоветовать две прекрасные книги:
  - первая для самых начинающих: Моли Брюс. UNIX/Linux: теория и практика программирования. Пер. с англ. – М.: КУДИЦ-ОБРАЗ, 2004;
  - вторая для более глубокого изучения: Митчелл Марк, Оулдем Джеффри, Самьюэл Алекс. Программирование для Linux. Профессиональный подход. — М.: Издательский дом "Вильямс", 2002 (эта книга также выложена на английском языке в Интернете по адресу **http://www.advancedlinuxprogramming.com**).
- Как уже было отмечено ранее, большая часть кода в этой книге относится к классу сетевых приложений, поэтому вам необходимо знать, как программировать сетевые приложения в среде Linux. Я не буду по ходу книги останавливаться и объяснять такие основополагающие сетевые функции, как `socket()`, `bind()`, `connect()`, `listen()`, `inet_aton()`, `htons()`, `sendto()`, `recvfrom()`, `setsockopt()`, `select()`, или такие структуры, как `sockaddr_in`, `sockaddr_ll` и пр. Я предполагаю, что даже если вы и не имеете опыта в области программирования сетевых приложений, то, по крайней мере, знакомы с этой темой на уровне чтения хорошей книги. Если же это не так, то настоятельно советую изучить классический труд: Стивенс У. Р. UNIX: разработка сетевых приложений. — СПб.: Питер, 2003.

Разумеется, это не все знания, которые вам потребуются для понимания такой всеобъемлющей книги, как эта. Например, при чтении некоторых глав вам понадобится знание ассемблера или опыт программирования загружае-

[1] Я набирал весь код для этой книги в текстовом редакторе, который вызывается комбинацией <Shift>+<F4> в файловом менеджере Midnight Commander.

мых модулей Linux и многое другое. Но не переживайте, в таких случаях я буду давать необходимые начальные сведения или источники, к которым следует обращаться.

Серия "Программирование боевого софта"

Эта книга является лишь первой книгой из серии, вторая книга будет называться "Программирование боевого софта под Windows". Не пропустите!

Связь с автором

Связаться со мной можно по одному из следующих адресов электронной почты: **sklyaroff@sklyaroff.ru**, **sklyaroff@mail.ru**, **sklyarov@real.xakep.ru**. Также я предлагаю вам посетить мой сайт: **http://www.sklyaroff.ru** (или **http://www.sklyaroff.com**).

ЧАСТЬ I

Инструментарий разработчика боевого софта

Глава 1

Основные инструменты

Подобно слесарю программист должен владеть некоторыми специализированными инструментами. Конечно, можно вытачивать все детали одним ржавым напильником, но, согласитесь, быстрее и эффективнее это делать с помощью токарного станка с программным управлением. А при программировании нестандартного хакерского программного обеспечения, без специальных инструментов просто не обойтись. Поэтому не случайно эта глава поставлена на первое место: прежде чем приступать к работе, научитесь пользоваться необходимым инструментарием!

Здесь собраны сведения только о стандартных утилитах, т. е. тех, которые идут в стандартной комплектации с любым полноценным дистрибутивом Linux. Этих инструментов обычно достаточно, чтобы решить весь спектр основных задач. Во второй главе приведен обзор дополнительных утилит, которые позволяют решить узкоспециализированные задачи.

Не следует в этих главах искать сведения о таких основополагающих программах, как ps, who, man, gcc и т. д. Если вы не умеете с ними работать, то вам явно рано читать эту книгу, поставьте ее на полку и изучите сначала литературу, которая предлагалась во введении.

Из нестандартных программ стоило бы, пожалуй, отметить только виртуальную машину VMWare. Это воистину уникальная и незаменимая программа для хакера! На сайте разработчика **http://www.vmware.com** можно приобрести версию этой виртуальной машины под Linux и Windows или скачать демо-версию. Первоначально я хотел посвятить VMWare отдельную главу, но потом передумал, так как реально по VMWare нужно писать отдельную книгу (что я и сделаю, возможно, в будущем). На самом деле это довольно простая в использовании программа, но чтобы по максимуму задействовать все ее возможности, нужны навыки администратора сети. Так как я владею такими навыками, то мне не составило труда с помощью VMWare создать

небольшую локальную Ethernet-сеть прямо на своем рабочем компьютере. Львиная доля сетевых программ из этой книги была написана и протестирована именно под VMWare.

1.1. Утилита gdb

Утилита gdb (*сокр. от* GNU Debugger) — это стандартный консольный отладчик в Linux и во многих других UNIX-системах. Существуют графические интерфейсы к gdb, например DDD, но мы рассматривать их не будем, т. к. они не являются стандартными средствами Linux и в мире UNIX не пользуются популярностью.

Существует три класса объектов, называемые целями (target), которые программист может отлаживать в gdb: *исполняемые файлы*, *дампы памяти* (core-файлы) и *процессы*. Напомню, что в core-файле находится образ памяти процесса, образованный, как правило, в результате его аварийного завершения. Посмотрим, как каждую из целей можно загрузить в gdb для отладки. Для этого gdb предоставляет множество различных способов. Во-первых, любую цель можно загрузить из командной строки одновременно с запуском gdb. Далее перечислены основные способы.

Способы загрузки исполняемого файла в gdb:

```
# gdb имя_программы
# gdb -exec имя_программы
# gdb -e имя_программы
```

Способы загрузки дампа памяти в gdb:

```
# gdb -core имя_core
# gdb -c имя_core
# gdb имя_программы имя_core
```

В последнем варианте первым аргументом должно указываться имя программы, которая сгенерировала данный core-файл.

Способы загрузки процесса в gdb:

```
# gdb -c pid_процесса
# gdb имя_процесса pid_процесса
```

PID любого процесса узнается с помощью команды `ps`.

Любую из целей можно загрузить в уже запущенном отладчике gdb.

Загружаем исполняемый файл:

```
(gdb) file имя_программы
(gdb) exec-file имя_программы
```


Загружаем дамп памяти:

```
(gdb) core-file имя_core
```

Загружаем процесс:

```
(gdb) attach pid_процесса
```

Когда вы закончите отлаживать процесс, его можно выгрузить из gdb командой `detach`. Отсоединенный процесс продолжает выполнение в системе, а вы можете присоединить другой процесс.

При запуске gdb выводит довольно внушительный объем данных о копирайте, подавить этот вывод можно, указав при запуске ключ `-q`.

Для более удобной и эффективной отладки ваша программа должна быть откомпилирована с отладочной информацией. За это в gcc отвечает ключ `-g`. Отладочная информация позволит вам видеть в gdb имена переменных, функций, номера строк и пр., как они были в исходном тексте вашей программы. Если отладочная информация недоступна, gdb будет отлаживать программу на уровне ассемблерных команд.

Когда программа загружена в отладчик, нужно установить точку останова. Существует 3 типа точек останова:

1. Breakpoints (обычная точка останова). Программа приостанавливается, когда выполнение доходит до определенного адреса или функции.
2. Watchpoints (точка наблюдения). Выполнение программы приостанавливается, когда программа обращается к определенной переменной (считывает или изменяет ее значение).
3. Catchpoints (точка перехвата). Приостановка происходит при определенном событии (например, получение сигнала).

Рассмотрим их подробнее.

Breakpoints. Обычные точки останова устанавливаются командой `break` (сокращенно `b`)[1]. Например, следующая команда устанавливает точку останова на функцию `main()`:

```
(gdb) break main
```

Можно установить точку останова и на произвольный адрес (перед адресом обязательно указывается символ "*"). Это может понадобиться в тех частях вашей программы, которые не имеют отладочной информации или исходных файлов. Пример:

```
(gdb) b *0x801b7000
```

Watchpoints. Существуют различные виды точек наблюдения, и задаются они различными командами. Команда `watch` (сокращенно `wa`) устанавливает

[1] Все основные команды в gdb имеют длинную и краткую формы.

точку наблюдения, которая остановит программу, когда значение указанной переменной изменится:

```
(gdb) wa переменная
```

Команда `rwatch` (сокращенно `rw`) устанавливает точку наблюдения, которая остановит программу, когда значение указанной переменной считывается программой.

```
(gdb) rw переменная
```

Команда `awatch` (сокращенно `aw`) устанавливает точку наблюдения, которая остановит программу, когда программа читает или записывает переменную:

```
(gdb) aw переменная
```

Catchpoints. Точка перехвата устанавливаются следующей командой:

```
(gdb) catch событие
```

Программа остановится, когда произойдет указанное *событие*. Событие может быть одним из: `throw` — происходит исключение Си++, `catch` — перехват исключения Си++, `exec` — вызов `exec()`, `fork` — вызов `fork()`, `vfork` — вызов `vfork()`.

Об остальных событиях можно узнать командой `help catch`. К сожалению, многие события еще не реализованы в gdb.

Информацию обо всех установленных точках останова можно вывести командой `info breakpoints` (сокращенно `i b`). Если какая-то точка останова не нужна, то ее можно сделать неактивной с помощью команды `disable`:

```
(gdb) disable b номер_точки
```

Обратно активировать точку останова можно командой `enable`:

```
(gdb) enable b номер_точки
```

Номер точки, а также ее статус (активна она или нет) показывается командой `i b`.

Если же точка останова не требуется вообще, то ее можно удалить следующей командой:

```
(gdb) delete breakpoint номер_точки
```

или короче:

```
(gdb) d b номер_точки
```

Команда `d` без аргументов удалит все точки останова.

После того как точка останова установлена, нужно запустить программу на выполнение командой `run` (`r`). Выполнение будет происходить, пока не встретится точка останова. Продолжить выполнение остановленной программы можно командой `continue` (`c`). Можно выполнять пошаговую трассировку командами: `step N` (`s N`) — выполнить *N* следующих строк кода с заходом

в функцию и `next N` (`n N`) — без захода в функцию. Если *N* не задано, то по умолчанию выполняется одна строка. Команды `stepi N` (`si N`) и `nexti N` (`ni N`) делают то же самое, но работают не со строками исходного текста, а с машинными командами. Команда `finish` (`fin`) выполняет программу до момента выхода из текущей функции.

Команда `print` (`p`) может вывести значение выражения на исходном языке (например, `p 2+3`), содержимое переменной (`p my_var`), содержимое регистра (`p $eax`) или ячейки памяти (`p *0x8018305`). Команда `x` позволяет просматривать сразу множество ячеек памяти. Формат команды — `x/Nfu адрес`, где:

- *адрес* — выражение, задающее адрес, с которого вы хотите начать отображение памяти (ставить символ звездочки перед адресом в этой команде не нужно);
- *N* — число повторений (по умолчанию 1), определяет, сколько памяти отобразить (считая в единицах `u`);
- *f* — формат отображения, может быть одним из следующих значений: `s` (строка, оканчивающаяся нулем), `i` (машинная инструкция), `x` (шестнадцатеричная форма — это первоначальное значение по умолчанию);
- *u* — размер единицы измерений, который может быть одним из следующих значений: `b` (байт), `h` (два байта), `w` (четыре байта — это первоначальное значение по умолчанию), `g` (восемь байт).

Если вы используете значения по умолчанию для *Nfu*, то не следует вводить косую черту "/". Пример команды, которая выводит 20 шестнадцатеричных слов, начиная с адреса 0x40057936:

```
(gdb) x/20xw 0x40057936
```

Команда `set` позволяет модифицировать содержимое регистров или ячеек памяти. Например, `set $ebx = 1` записывает в регистр `ebx` единицу.

Команда `info registers` (сокращенно `i r`) показывает содержимое всех регистров. Если нужно посмотреть конкретные регистры, то их нужно указать сразу за этой командой, например:

```
(gdb) i r ebp eip
```

Команда `info share` выводит сведения о разделяемых библиотеках, загруженных в данный момент. Команды `info frame`, `info args`, `info local` показывают содержимое текущего кадра стека, аргументы функции и локальные переменные. Команда `backtrace` (сокращенно `bt`) выводит кадр стека для каждой активной подпрограммы. Выход из отладчика осуществляется командой `quit` (`q`). С помощью команды `help` (`h`) можно узнать подробности по любой команде.

1.2. Утилита ifconfig

Утилита ifconfig используется для конфигурирования сетевых интерфейсов и позволяет изменить такие параметры интерфейса, как IP-адрес, маску сети, MAC-адрес и пр. Для программиста эта утилита в первую очередь полезна той информацией, которую она выдает с помощью параметра `-a`:

```
# ifconfig -a
eth0    Link encap:Ethernet  HWaddr 00:0C:29:DE:7A:BC
        inet addr:192.168.10.130  Bcast:192.168.10.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1443845 errors:0 dropped:0 overruns:0 frame:0
        TX packets:3419238 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
        Interrupt:10 Base address:0x10a4

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:1447064 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1447064 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
```

Сначала выведены данные об Ethernet-интерфейсе `eth0`, а потом — об интерфейсе обратной связи `lo`. Если команду ifconfig выполнить без параметров, то она не покажет отключенные интерфейсы (если таковые были) опцией `down` (см. ниже).

В выведенных данных мы можем увидеть: IP-адрес, назначенный интерфейсу (`inet addr`), широковещательный адрес (`Bcast`), сетевую маску (`Mask`), MAC-адрес (`HWaddr`) и максимальную единицу передачи данных (`MTU`) в байтах. Также могут представлять интерес строки, показывающие число успешно переданных (`TX packets`) или полученных (`RX packets`) пакетов, а также ошибочных (`errors`), отброшенных (`dropped`) и повторенных (`overruns`). Метка `collisions` показывает число коллизий в сети, а `txqueuelen` — длину очереди передачи для устройства. Также можно узнать номер аппаратного прерывания IRQ (`Interrupt`), используемого устройством.

Если в командной строке указать имя интерфейса, то будут выведены данные только по этому интерфейсу:

```
# ifconfig eth0
```

С помощью опции `mtu N` можно установить MTU (Maximum Transition Unit, максимальная единица передачи) для интерфейса:

```
# ifconfig eth0 mtu 1000
```


Утилита ifconfig не позволит установить MTU больше максимально допустимого значения, которое для Ethernet составляет 1500 байт. Опция `-arp` (с минусом) позволяет отключить использование протокола ARP для указанного интерфейса, а опция `arp` (без минуса) — включить обратно.

```
# ifconfig eth0 -arp
# ifconfig eth0
eth0    Link encap:Ethernet  HWaddr 00:0C:29:DE:7A:BC
        inet addr:192.168.10.130  Bcast:192.168.10.255  Mask:255.255.255.0
        UP BROADCAST RUNNING NOARP MULTICAST  MTU:1500  Metric:1
        ...
```

Опция `promisc` (без минуса) включает *неразборчивый режим* (promiscuous mode) для интерфейса. В этом режиме все проходящие по сети пакеты будут приниматься интерфейсом. Такой режим обычно используют снифферы в своей работе *(см. гл. 9)*. Опция `-promisc` (с минусом) отключает неразборчивый режим.

```
# ifconfig eth0 promisc
# ifconfig eth0
eth0    Link encap:Ethernet  HWaddr 00:0C:29:DE:7A:BC
        inet addr:192.168.10.130  Bcast:192.168.10.255  Mask:255.255.255.0
        UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
        ...
```

Для назначения (изменения) IP-адреса интерфейса используется опция `inet`, а для маски опция `netmask`. Например:

```
# ifconfig eth0 inet 200.168.10.15 netmask 255.255.255.192
# ifconfig eth0
eth0  Link encap:Ethernet  HWaddr 00:0C:29:DE:7A:BC
      inet addr:200.168.10.15  Bcast:200.168.10.255  Mask:255.255.255.192
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      ...
```

С помощью опции `down` можно отключить интерфейс, а опцией `up` включить обратно:

```
# ifconfig eth0 down
# ifconfig eth0 up
```

Опция `hw class address` изменяет аппаратный адрес (MAC-адрес) соответствующего интерфейса, если драйвер устройства поддерживает такую возможность. После ключевого слова `hw` необходимо указать имя класса оборудования, а также MAC-адрес в текстовом виде. В настоящее время поддерживается оборудование классов ether (Ethernet), ax25 (AMPR AX.25), ARCnet и netrom

(AMPR NET/ROM). Перед изменением аппаратного адреса необходимо отключить интерфейс опцией down. Ниже приведен пример изменения MAC-адреса на интерфейсе eth0:

```
# ifconfig eth0 down
# ifconfig eth0 hw ether 13:13:13:13:13:13
# ifconfig eth0 up
# ifconfig eth0
eth0    Link encap:Ethernet  HWaddr 13:13:13:13:13:13
        inet addr:192.168.10.130  Bcast:192.168.10.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        ...
```

Утилита ifconfig позволяет назначить одному интерфейсу множество IP-адресов-псевдонимов. Ниже показан пример назначения трех IP-адресов одному интерфейсу eth0 (адреса-псевдонимы следует выбирать из той же подсети, что и базовый адрес 192.168.10.130):

```
# ifconfig eth0:0 192.168.10.200
# ifconfig eth0:1 192.168.10.201
# ifconfig eth0:2 192.168.10.202
# ifconfig -a
eth0    Link encap:Ethernet  HWaddr 00:0C:29:DE:7A:BC
        inet addr:192.168.10.130  Bcast:192.168.10.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1469698 errors:0 dropped:0 overruns:0 frame:0
        TX packets:3440721 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
        Interrupt:10 Base address:0x10a4

eth0:0 Link encap:Ethernet  HWaddr 00:0C:29:DE:7A:BC
        inet addr:192.168.10.200  Bcast:192.168.10.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        Interrupt:10 Base address:0x10a4

eth0:1 Link encap:Ethernet  HWaddr 00:0C:29:DE:7A:BC
        inet addr:192.168.10.201  Bcast:192.168.10.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        Interrupt:10 Base address:0x10a4

eth0:2 Link encap:Ethernet  HWaddr 00:0C:29:DE:7A:BC
```



```
        inet addr:192.168.10.202  Bcast:192.168.10.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        Interrupt:10 Base address:0x10a4
        ...
```

Теперь к интерфейсу можно обращаться по любому из четырех IP-адресов: 192.168.10.130, .200, .201, .202. Эта возможность часто используется администраторами для создания виртуальных web-узлов на базе IP-адресов. Удалить псевдоним можно с помощью параметра down, например:

```
# ifconfig eth0:1 down
```

1.3. Утилита netstat

Утилита netstat выводит различную информацию, касающуюся функционирования сети. Если netstat вызвать без параметров, то она выдаст информацию об установленных соединениях и малоинтересные сведения о внутренних очередях и файлах, которые используются для межпроцессного взаимодействия. Порты, находящиеся в состоянии ожидания запросов на установление соединения по умолчанию не отображаются. Вывести список *всех* открытых портов позволяет параметр -a. Ниже показан пример такого вывода:

```
# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:1024                  *:*                     LISTEN
tcp        0      0 *:sunrpc                *:*                     LISTEN
tcp        0      0 *:ftp                   *:*                     LISTEN
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 *:telnet                *:*                     LISTEN
tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
tcp        0      0 192.168.10.130:ssh      192.168.10.128:39806
ESTABLISHED
udp        0      0 *:1024                  *:*
udp        0      0 *:686                   *:*
udp        0      0 *:sunrpc                *:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node Path
unix  2      [ ACC ]     STREAM     LISTENING     1581   /dev/gpmctl
unix  2      [ ACC ]     STREAM     LISTENING     939    /var/run/pump.sock
unix  13     [ ]         DGRAM                    1178   /dev/log
unix  2      [ ACC ]     STREAM     LISTENING     1617   /tmp/.font-unix/fs7100
```

```
unix  2      [ ]         DGRAM                    690847
unix  2      [ ]         DGRAM                    252658
unix  2      [ ]         DGRAM                    12241
unix  2      [ ]         DGRAM                    1673
unix  2      [ ]         DGRAM                    1620
unix  2      [ ]         DGRAM                    1584
unix  2      [ ]         DGRAM                    1556
unix  2      [ ]         DGRAM                    1439
unix  2      [ ]         DGRAM                    1413
unix  2      [ ]         DGRAM                    1223
unix  2      [ ]         DGRAM                    1187
unix  2      [ ]         STREAM     CONNECTED     730
```

При нефункционирующей службе DNS netstat будет выводить информацию на экран с большими задержками. Чтобы этого избежать, следует дополнительно указывать флаг `-n`, который не позволяет утилите выполнять преобразование IP-адресов в доменные имена:

```
# netstat -an
```

При этом все адреса будут отображены в числовом формате.

Из примера вывода выше видно, что вся выходная информация утилиты netstat поделена на две части. В первой части (**Active Internet connections**) показаны все установленные соединения и ожидающие подключения порты. В колонке **Proto** указывается протокол (TCP или UDP), по которому работает соединение или служба. В колонках **Recv-Q** и **Send-Q** отображается число байтов, соответственно, в буферах чтения и записи сокетов. В колонках **Local Address** и **Foreign Address** указан соответственно локальный и удаленный адрес.

Звездочкой ("*") обычно отображается локальный адрес и порт (если был указан параметр `-n`, то локальный адрес показывается нулями: 0.0.0.0). Адреса представлены в формате *имя_компьютера_или_ip_адрес:служба*, где *служба* — номер порта или имя для известных служб (соответствия между номерами портов и именами служб определены в файле /etc/services)[2].

В колонке **State** показано состояние соединения. Наиболее распространенными состояниями являются: ESTABLISHED (установлено) — для активных соединений; LISTEN (прослушивание) — для сервисов, ожидающих поступления запросов (при отсутствии опции `-a` не показываются); TIME_WAIT (ожидание закрытия) — для соединений, находящихся в процессе закрытия.

[2] В некоторых версиях UNIX в выводе утилиты netstat для отделения имени службы или номера порта от имени компьютера или его IP-адреса используется не двоеточие, а обычная точка.

Состояние соединения имеет смысл только для протокола TCP. Протокол UDP не проверяет факт установления соединения.

Таким образом, из примера видно, что на локальном узле имеется множество прослушиваемых портов и одно входящее соединение по протоколу SSH с удаленного адреса 192.168.10.128:39806.

Во второй части выходной информации (**Active UNIX domain sockets**) отображены внутренние очереди и файлы, которые используются для межпроцессного взаимодействия.

С помощью параметра `-t` можно затребовать вывод только сведений о TCP-портах:

```
# netstat -tan
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:1024            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 192.168.10.130:22       192.168.10.128:58291    ESTABLISHED
```

Соответственно с помощью параметра `-u` можно вывести сведения только о UDP-портах:

```
# netstat -uan
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 0.0.0.0:1024            0.0.0.0:*
udp        0      0 0.0.0.0:686             0.0.0.0:*
udp        0      0 0.0.0.0:111             0.0.0.0:*
```

Параметр `-i` позволяет вывести информацию о состоянии сетевых интерфейсов:

```
# netstat -i
Kernel Interface table
Iface   MTU Met    RX-OK RX-ERR RX DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500   0  1428232      0      0      0  3418346      0      0      0 BMRU
lo    16436   0  1446930      0      0      0  1446930      0      0      0 LRU
```

Во многом эта информация повторяет информацию, выдаваемую командой `ifconfig -a`. В колонках, названия которых начинаются с приставки RX-, показано: число успешно принятых, ошибочных, отброшенных и повторенных пакетов при получении (от англ. *received* — получено). В колонках, начинающихся с приставки TX-, показано число успешно принятых, ошибочных,

отброшенных и повторенных пакетов при отправке (от англ. *transmitted* — передано).

Netstat позволяет отслеживать работу сетевых интерфейсов в реальном времени. Для этого необходимо дополнительно указать параметр -c, который заставляет netstat выдавать на экран статистику с интервалом в одну секунду:

```
# netstat -i -c
```

Это может помочь в выслеживании источников ошибок в сети.

Параметр -s позволяет посмотреть статистику функционирования различных сетевых протоколов:

```
# netstat -s
Ip:
    2869242 total packets received
    2 with invalid headers
    0 forwarded
    37 incoming packets discarded
    1489607 incoming packets delivered
    4865030 requests sent out
    38 fragments dropped after timeout
    174870 reassemblies required
    87357 packets reassembled ok
    38 packet reassembles failed
    193194 fragments created
Icmp:
    478041 ICMP messages received
    515 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 9559
        timeout in transit: 74
        echo requests: 177230
        echo replies: 291178
    177978 ICMP messages sent
    0 ICMP messages failed
...
```

Параметр -r позволяет вывести таблицу маршрутизации ядра:

```
# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.10.0    *               255.255.255.0   U        40 0          0 eth0
127.0.0.0       *               255.0.0.0       U        40 0          0 lo
```


В Linux программа netstat имеет параметр -p, который отображает какой процесс связан с конкретным портом:

```
# netstat -anp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program
name
tcp        0      0 0.0.0.0:1024            0.0.0.0:*               LISTEN      510/rpc.statd
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      495/portmap
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN      742/xinetd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      722/sshd
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN      742/xinetd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      782/sendmail:
                                                                                 accep
tcp        0      0 192.168.10.130:22       192.168.10.128:39806    ESTABLISHED 9989/sshd
udp        0      0 0.0.0.0:1024            0.0.0.0:*                           510/rpc.statd
udp        0      0 0.0.0.0:686             0.0.0.0:*                           510/rpc.statd
udp        0      0 192.168.10.130:1129     192.168.10.1:53         ESTABLISHED
                                                                                 10058/tcpdump
udp        0      0 0.0.0.0:111             0.0.0.0:*                           495/portmap
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node PID/Program name    Path
unix  2      [ ACC ]     STREAM     LISTENING     1581   795/gpm             /dev/gpmctl
unix  2      [ ACC ]     STREAM     LISTENING     939    415/pump
                                                                            /var/run/pump.sock
unix  13     [ ]         DGRAM                    1178   476/syslogd
                                                                            /dev/log
unix  2      [ ACC ]     STREAM     LISTENING     1617   853/xfs
                                                                            /tmp/.font-unix/fs7100
unix  2      [ ]         DGRAM                    690847 880/login -- root
unix  2      [ ]         DGRAM                    252658 742/xinetd
unix  2      [ ]         DGRAM                    12241  879/login -- root
unix  2      [ ]         DGRAM                    1673   878/login -- root
unix  2      [ ]         DGRAM                    1620   853/xfs
unix  2      [ ]         DGRAM                    1584   807/crond
unix  2      [ ]         DGRAM                    1556   782/sendmail: accep
unix  2      [ ]         DGRAM                    1439   695/automount
unix  2      [ ]         DGRAM                    1413   646/apmd
```

```
unix  2      [ ]         DGRAM                    1223   510/rpc.statd
unix  2      [ ]         DGRAM                    1187   481/klogd
unix  2      [ ]         STREAM     CONNECTED     730    1/init [3]
```

Как видите, по сравнению с первым листингом добавилась еще одна седьмая колонка **PID/Program name**, в которой показан идентификатор процесса (PID) с именем сервиса. Из-за недостатка места на странице информация из этой колонки перенесена на следующую строку. Стоит заметить, что не во всех версиях UNIX утилита netstat имеет параметр `-p`, поэтому в таких системах функцию этого параметра выполняет утилита lsof.

1.4. Утилита lsof

Утилита lsof входит в состав большинства современных ОС Linux. Если в вашей системе lsof отсутствует, то ее можно скачать по адресу: **ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/**.

Название lsof является сокращением от англ. *list open files* — перечислить открытые файлы. Поэтому lsof, запущенная без параметров, покажет все открытые файлы, каталоги, библиотеки, потоки UNIX, открытые порты и процессы, открывшие их. Параметр `-i` выводит только открытые порты и соответствующие процессы, открывшие их:

```
# lsof -i
COMMAND    PID USER   FD   TYPE DEVICE SIZE NODE NAME
portmap    495 root    3u  IPv4   1211       UDP *:sunrpc
portmap    495 root    4u  IPv4   1212       TCP *:sunrpc (LISTEN)
rpc.statd  510 root    4u  IPv4   1232       UDP *:686
rpc.statd  510 root    5u  IPv4   1241       UDP *:1024
rpc.statd  510 root    6u  IPv4   1244       TCP *:1024 (LISTEN)
sshd       722 root    3u  IPv4   1482       TCP *:ssh (LISTEN)
xinetd     742 root    3u  IPv4   1509       TCP *:ftp (LISTEN)
xinetd     742 root    4u  IPv4   1510       TCP *:telnet (LISTEN)
sendmail   782 root    4u  IPv4   1557       TCP localhost.localdomain:smtp (LISTEN)
```

Из этой информации можно узнать, что службы ftp и telnet запускаются с помощью суперсервера xinetd, а, например, служба smtp запускается с помощью sendmail, поэтому ее нельзя отключить изменением конфигурационного файла `/etc/xinetd.conf`.

Можно также выводить информацию только по конкретному сервису:

```
# lsof -i TCP:ftp
COMMAND PID USER   FD   TYPE DEVICE SIZE NODE NAME
xinetd  742 root    3u  IPv4   1509       TCP *:ftp (LISTEN)
```


1.5. Утилита tcpdump

Утилита tcpdump — это сетевой анализатор пакетов, разработанный Lawrence Berkeley National Laboratory. Официальная страница этой утилиты: **http://www.tcpdump.org**. Во время написания мной сетевых утилит для этой книги tcpdump в моей системе использовалась практически постоянно.

1.5.1. Опции командной строки

Если tcpdump запустить без каких-либо параметров, то она будет перехватывать все сетевые пакеты и выводить о них информацию. С помощью параметра `-i` можно указать сетевой интерфейс, с которого следует принимать данные:

```
# tcpdump -i eth2
```

Если требуются лишь пакеты, получаемые или отправляемые от определенного узла, то его имя или IP-адрес нужно указать после ключевого слова `host`:

```
# tcpdump host namesrv
```

Если же нужны пакеты, которыми обмениваются, например, узлы namesrv1 и namesrv2, то можно использовать такой фильтр:

```
# tcpdump host namesrv1 and host namesrv2
```

Или сокращенно:

```
# tcpdump host namesrv1 and namesrv2
```

Для отслеживания только исходящих пакетов от какого-либо узла нужно указать два ключевых слова `src host`:

```
# tcpdump src host namesrv
```

А для отслеживания только входящих пакетов нужно указать `dst host`:

```
# tcpdump dst host namesrv
```

Ключевые слова `src port` и `dst port` позволяют указать, соответственно, порт отправителя и порт получателя, например:

```
# tcpdump dst port 513
```

Если нужно отслеживать один из трех протоколов TCP, UDP, ICMP, то его имя можно просто указать в командной строке. С помощью булевых операторов `and` (или `&&`), `or` (или `||`) и `not` (или `!`) можно задавать фильтры произвольной сложности. Ниже приведен пример фильтра, отслеживающего только ICMP-пакеты, приходящие из внешней сети:

```
# tcpdump icmp and not src net localnet
```

Можно проверять конкретные биты или байты в заголовках протоколов, для чего используется такой формат: `proto[expr:size]`, где `proto` — один из протоколов `ether`, `fddi`, `tr`, `ip`, `arp`, `rarp`, `tcp`, `udp`, `icmp` или `ip6`; `expr` — смещение в байтах от начала заголовка пакета; `size` — дополнительное поле, указывающее сколько байт рассматривать (может отсутствовать, тогда рассматривается 1 байт). Например, чтобы отбирать только TCP-сегменты, в которых установлен флаг `SYN`, следует использовать такой фильтр:

```
# tcpdump 'tcp[ 13 ]==2'
```

Здесь просто нужно знать, что с 13 байта заголовка TCP расположены 8 бит флагов (1 байт), где `SYN` является вторым битом по номеру *(см. разд. 3.4.4)*. Так как он должен быть установлен в 1, то байт флагов в двоичном виде будет выглядеть как 00000010. С помощью параметра `-c` можно указать количество принимаемых пакетов:

```
# tcpdump -c 10
```

В итоге будет получено всего 10 пакетов. Параметр `-a` позволяет отображать IP-адреса в символьном виде, если это возможно (правда это сильно замедляет работу утилиты):

```
# tcpdump -a
```

С помощью флага `-vvv` можно выводить много-много информации (от англ. *very very verbose*). Меньшую информацию выдают флаги: `-v` и `-vv`.

1.5.2. Формат выходной информации tcpdump

В начале каждой строки листинга tcpdump ставится отметка времени, которая является текущим временем часов в формате: `hh:mm:ss.frac`, где `frac` — это доли секунд. За отметкой времени может указываться интерфейс, на который происходит прием пакетов, например, eth0, eth1, lo и т. п. Запись `eth0<` означает, что идет прием пакетов на интерфейс eth0. Соответственно запись `eth0>` означает, что идет отправка пакетов в сеть с интерфейса eth0. Дальнейшие сведения зависят от типа принимаемого пакета (ARP/RARP, TCP, UDP, NBP, ATP и т. д.). Далее приведены форматы для некоторых основных типов пакетов.

TCP Packets (пакеты TCP):

```
Src.port > dst.port: flags data-seqno ack window urgent options
```

где:

- `Src.port` и `dst.port` — это IP-адрес и порт, соответственно, источника и приемника пакетов;
- `Flags` — это флаги, установленные в заголовке TCP-пакета. Могут принимать значения: `S` (`SYN`), `F` (`FIN`), `P` (`PUSH`), `R` (`RST`); также в этом поле может стоять одна точка ".", которая означает отсутствие установленных флагов;

- `Data-seqno` — описывает данные, содержащиеся в пакете, в таком формате: *first:last*(*nbytes*), где *first* и *last* — номер последовательности первого и последнего байта пакета, *nbytes* — количество байтов данных. Если параметр *nbytes* равен нулю, то *first* и *last* совпадают;
- `Ack` — следующий номер последовательности (ISN + 1);
- `Window` — размер окна;
- `Urgent` — указывает на наличие срочных данных в пакете (флаг URG);
- `Options` — здесь могут указываться дополнительные сведения, например, `<mss 1024>` (максимальный размер сегмента).

UDP Packets (пакеты UDP):

```
Src.port > dst.port: udp nbytes
```

где:

- `Udp` — метка, указывающая на то, что идет анализ UDP-пакетов;
- `Nbytes` — число байтов данных, которые содержит UDP-пакет.

ICMP Packets (пакеты ICMP):

```
Src > dst: icmp: type
```

где:

- `Icmp` — метка, идентифицирующая ICMP-пакет;
- `Type` — тип ICMP-сообщения, например, echo request или echo reply.

Глава 2

Дополнительные инструменты

Утилиты, описанные в этой главе, требуются программисту не так часто, но в некоторых ситуациях они просто незаменимы, поэтому иметь общее представление и помнить о них программисту необходимо. Все они, как правило, входят в штатную поставку любой полноценной операционной системы Linux. Многие из описываемых утилит входят также в пакет GNU Binutils, который является фундаментальной частью системы Linux. Сайт разработчиков пакета Binutils расположен по адресу: **http://sources.redhat.com/binutils/**. В этой главе приведены только общие обзорные сведения, подробности по каждой утилите смотрите в man.

2.1. Утилита time

Утилита time показывает время, затраченное на выполнение программы. Пример:

```
# time ./your_prog
real 0m0.008s
user 0m0.001s
sys 0m0.010s
```

Здесь `real` — это астрономическое время, в течение которого выполнялась программа; `user` — время центрального процессора, потраченное на исполнение программы; `sys` — время, затраченное на программу операционной системой. Понятно, что буква `m` указывает минуты, а `s` — секунды в десятичных дробях. Если нужно отследить время выполнения программы, которая использует множество аргументов командной строки и/или каналы, то утилиту time следует использовать так:

```
# time /bin/sh -c "your_prog -flags|my_prog"
```


2.2. Утилита gprof

Утилита gprof — это профайлер (профилировщик). С помощью профайлера можно установить, какие функции в программе вызываются чаще, чем нужно, и какие из них затрачивают больше всех вычислительных ресурсов, т. е. можно выявить в программе "узкие места".

Воспользоваться gprof просто. Сначала компилируется и компонуется программа с опциями профилирования (для компилятора gcc должен быть указан флаг `-pg`). Затем программа запускается, в результате чего генерируются профильные данные и записываются в файл gmon.out. Профильный файл не создастся, если программа завершается аварийно, т. е. ваша программа должна быть уже отлажена. На последнем этапе запускается сама gprof, которой нужно передать имя исполняемого файла.

Утилита gprof анализирует файл gmon.out и выдает информацию о том, сколько времени заняло выполнение каждой функции. В общем случае информация будет состоять из двух таблиц: "Простой профиль" ("Flat profile") и "Граф вызовов" ("Call graph") с замечаниями, кратко объясняющими их содержимое. Простой профиль показывает, сколько времени выполнялась каждая функция и сколько раз она вызывалась. По нему легко установить, какие функции программы затрачивают больше всего времени. Граф вызовов может подсказать те места, в которых можно попытаться исключить вызовы функций, требующие много времени на выполнение.

В таблице графа вызовов показано для каждой функции, какие функции ее вызывали, какие функции вызывала она сама и сколько раз. Также здесь есть информация, сколько времени было затрачено на выполнение подпрограмм в каждой функции. Если утилиту gprof выполнить с опцией `-A`, то будет отображен исходный текст программы с процентными показателями времени выполнения. Профилировку имеет смысл делать только в больших программах с множеством вызовов функций. Пример использования:

```
# gcc -pg -o your_prog your_prog.c
# ./your_prog
# gprof ./your_prog
```

2.3. Утилита ctags

Если программа состоит из множества модулей, которые в свою очередь разбросаны по множеству исходных файлов, то становится сложно отыскать определение нужной функции. Именно для быстрого поиска функций и предназначена утилита ctags. Достаточно ей передать исходные файлы вашей

программы, как она сформирует особый информационный файл с именем tags, состоящий из трех колонок, где в первой колонке будут названия всех функций, во второй — имена исходных файлов, в которых расположены эти функции, а в третьей — готовый шаблон для поиска функций по файловой системе с помощью таких утилит, как find. Пример содержимого файла tags:

```
main /usr/src/you_prog.c /^main()$/
func1 /usr/src/you_prog.c /^func1(arg1,arg2)$/
func2 /usr/src/you_prog.c /^func2(arg1,arg2)$/
```

Пример использования утилиты ctags:

```
# ctags *.c
```

2.4. Утилита strace

Утилита strace отслеживает все запрашиваемые вызовы и получаемые системные сигналы заданной программы. Используется просто:

```
# strace ./your_prog
```

Каждая строка выводимой информации будет соответствовать одному системному вызову. Где сначала будет указано имя системного вызова со списком аргументов (с сокращениями), а после знака равно возвращаемое значение. В выражении вида:

```
execve("./your_prog", ["./your_prog"], [/* 27 vars */]) = 0
```

`[/* 27 vars */]` — означает, что здесь идет список переменных среды (27 штук), которые опущены strace для краткости.

Strace, запущенная с параметром `-f`, позволит трассировать все процессы-потомки, созданные текущим процессом во время выполнения.

2.5. Утилита ltrace

Утилита ltrace подобна strace, но отслеживает все обращения указанной программы к динамическим библиотекам.

2.6. Утилита mtrace

Если ваша программа использует динамическую память, то очень желательно протестировать ее с помощью утилиты mtrace. mtrace отслеживает соответствие числа операций выделения и освобождения памяти, т. е. вылавливает утечки памяти.

Утечки памяти ведут к постепенному сокращению ресурсов системы, до полного их исчерпания. Чтобы выловить все возможные утечки памяти в вашей программе, придется проделать несколько шагов. Во-первых, нужно включить в программу файл `<mcheck.h>` и разместить в самом начале программы вызов функции `mtrace()`. Затем нужно указать имя файла, в котором будет сохраняться информация о проверке, делается это через экспортирование в переменную окружения, например:

```
# export MALLOC_TRACE=mem.log
```

После чего нужно запустить программу и все операции выделения и освобождения памяти будут регистрироваться в файле `mem.log`. Последним этапом вызывается утилита mtrace в следующем виде:

```
# mtrace you_prog $MALLOC_TRACE
```

Теперь нужно внимательно читать полученную информацию с указанием строк, где память не была освобождена. Чтобы полученные данные были записаны в файл, программа должна завершаться нормальным образом.

2.7. Утилиты make/gmake

Когда проект состоит из множества файлов, то любое изменение в одном из них неизбежно влечет за собой перекомпиляцию всех остальных, облегчить эту задачу способна утилита make (в некоторых системах она называется gmake). Этой утилите нужно передать простой текстовый файл под названием Makefile, который содержит информацию о правилах сборки и зависимостях. Правила записываются в следующем виде:

```
<цель>: <зависимости>
 <команда>
 <команда>
 ...
```

Первая цель в Makefile выполняется по умолчанию при запуске make без аргументов. Ее принято называть `all`, что эквивалентно команде `make all`. Пример Makefile:

```
all: your_prog

your_prog: your_prog.o foo.o boo.o
 gcc your_prog.o foo.o boo.o -o your_prog

your_prog.o: your_prog.c your_prog.h

foo.o: foo.c foo.h
```

```
boo.o: boo.c boo.h

clean:
 rm -f *.o you_prog
```

Цель `clean` предназначена для удаления всех сгенерированных объектных файлов и программ, чтобы make могла создать их заново. Чтобы собрать проект, достаточно в командной строке набрать:

```
# make
```

2.8. Утилиты automake/autoconf

Но есть еще один более простой способ создания Make-файлов с помощью стандартных утилит automake и autoconf. Сначала нужно подготовить файл `Makefile.am`, например:

```
bin_PROGRAMS = your_prog
you_prog_SOURCES = your_prog.c foo.c boo.c
AUTOMAKE_OPTIONS = foreign
```

Последняя опция указывает на то, что в проект не будут включаться файлы стандартной документации: NEWS, README, AUTHORS и ChangeLog. Но согласно стандарту их присутствие в GNU-пакете обязательно.

Теперь нужно создать файл `configure.in`. Это можно сделать с помощью утилиты autoscan. autoscan выполняет анализ дерева исходных текстов, корень которого указан в командной строке или совпадает с текущим каталогом, и создает файл `configure.scan`. Нужно просмотреть `configure.scan`, внести необходимые коррективы и затем переименовать в `configure.in`. И последним этапом следует запустить утилиты в следующем порядке:

```
# aclocal
# autoconf
# automake -a -c
```

В результате в текущей директории появятся сценарии `configure`, `Makefile.in` и файлы документации. Чтобы собрать проект, достаточно ввести следующие команды:

```
# ./configure
# make
```


2.9. Утилита ldd

Утилита ldd показывает все динамические библиотеки, от которых зависит программа. Пример использования:

```
# ldd ./your_prog
```

2.10. Утилита objdump

Утилита objdump позволяет исследовать файл. Например, она может дизассемблировать указанную программу (опция `-D`), показать все заголовки программы, в т. ч. файловые, секций и пр. (опция `-x`). Также может показать содержимое всех секций (опция `-s`), динамически перемещаемые данные (опция `-R`) и многое другое. Пример использования:

```
# objdump -D ./your_prog
```

2.11. Утилиты hexdump и od

Утилита hexdump может вывести содержимое указанного файла в десятичном виде (опция `-d`), шестнадцатеричном (опция `-x`), восьмеричном (опция `-b`) и в ASCII-символах (опция `-c`). Пример использования:

```
# hexdump -c ./your_prog
```

Утилита od аналогична утилите hexdump:

```
# od -c ./your_prog
```

2.12. Утилита strings

Утилита strings выводит последовательности ASCII-символов длиннее четырех (по умолчанию), которые хранятся в открытом виде в уже скомпилированной программе. Пример использования:

```
# strings ./your_prog
```

2.13. Утилита readelf

Утилита readelf позволяет исследовать организацию ELF-файла. Выводит файловые заголовки, заголовки секций и прочие структуры (*см. главу 15*).

2.14. Утилита size

Утилита size показывает размеры секций программы. По умолчанию только секции команд (.text), данных (.data) и секции неинициализированных данных (.bss) с общей суммой размеров всех этих секций в десятичном и шестнадцатеричном формате. Флаг `-A` позволяет вывести размеры всех имеющихся секций в файле.

```
# size ./your_prog
```

2.15. Утилита nm

Утилита nm выдает на стандартный вывод таблицу внешних символов для каждого файла, указанного в командной строке. Таблица символов используется для отладки приложения. Для каждого символа будет выведено его имя и указано, является ли он символом данных (переменной) или программным символом (меткой или именем функции) и пр. Пример:

```
# nm ./your_prog
```

2.16. Утилита strip

Когда программа отлажена, из нее можно удалить таблицу символов, для чего используется утилита strip. Пример:

```
# strip ./your_prog
```

2.17. Утилита file

Утилита file с помощью серии тестов, пытается классифицировать указанные файлы. Если файл текстовый, команда по первым 512 байтам пытается угадать язык программирования. Для исполняемых файлов file выводит информацию о платформе, версии и структуре библиотек файла. Примеры:

```
# file /bin/cat
/bin/cat: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically
linked (uses shared libs), stripped
# file ./code.c
./code.c: ASCII C program text, with CRLF, CR, LF line terminators
```

При запуске file нужно обязательно указывать путь к тестируемому файлу. Но этого можно избежать с помощью обратных кавычек и утилиты which,

которая сама определит путь к тестируемому файлу. В примере ниже путь к файлу `as` определяется автоматически:

```
# file `which as`
/usr/bin/as: ELF 32-bit LSB executable, Intel 80386, version 1,
dynamically linked (uses shared libs), stripped
```

2.18. Утилиты ipcs и ipcrm

Если ваша программа использует взаимодействие процессов, то вам могут пригодиться утилиты ipcs и ipcrm. Команда ipcs, указанная с флагом `-m`, показывает сведения о совместно используемых сегментах:

```
# ipcs -m
```

Если указать флаг `-s`, то ipcs покажет информацию о существующих группах семафоров. Утилита ipcrm позволяет удалить определенный сегмент в памяти или группу семафоров, например:

```
# ipcrm shm 2345097
```

удаляет сегмент под id равным 2345097.

Чтобы можно было работать с утилитами ipcs и ipcrm, в ядре должны быть включены опции:

- `option SYSVMSG` — поддержка сообщений в соответствии с System V;
- `option SYSVSEM` — поддержка семафоров в соответствии с System V;
- `option SYSVSHM` — возможность работы с разделяемой памятью в соответствии с System V.

2.19. Утилиты ar и ranlib

В пакете Binutils содержится архиватор ar, который используется для создания статических библиотек. Например:

```
# ar cr libmy.a file1.o file2.o
```

Флаги `cr` указывают на то, что должен быть создан архив, есть и другие флаги, например, для модификации или извлечения из архива (*см. man ar*). Для подключения полученной статической библиотеки к программам с помощью gcc или g++ нужно использовать флаг `-L`, который указывает, в каком каталоге следует искать библиотеку. Флаг `-L.` (с точкой) указывает на то, что библиотека находится в текущем каталоге. Затем все необходимые библиотеки перечисляются с помощью ключа `-l`, за которым указывается название библиотеки без префикса `lib` и без окончания `.a`, например:

```
# gcc -o your_prog.c -L. -lmy -o your_prog
```

Это работает в большинстве случаев, однако на некоторых системах получить статическую библиотеку не получится таким способом, т. к. после того как архиватор ar создаст архив, нужно в него добавить таблицу символов, т. е. список вложенных в библиотеку функций и переменных, чтобы компоновка проходила нормально. Делается это с помощью стандартной утилиты ranlib из пакета Binutils:

```
# ranlib libmy.a
```

После этого библиотеку можно подключать к программе с помощью gcc, как в предыдущем примере. Для получения статической библиотеки рекомендуется обрабатывать архив утилитой ranlib всегда.

2.20. Утилита arp

Утилита arp позволяет получить доступ к системному кэшу ARP.

Параметр `-a` выводит на экран все содержимое кэша ARP в BSD-стиле, а параметр `-e` в стиле Linux:

```
# arp -e
```

С помощью параметра `-d` можно удалить сведения об указанном адресе:

```
# arp -d IP-адрес
```

При этом запись из кэша не удаляется, а просто очищается поле аппаратного адреса (`HWaddress`).

Внести вручную MAC-адрес для указанного IP-адреса можно следующим образом:

```
# arp -s IP-адрес MAC-адрес
```

ЧАСТЬ II

Сетевой боевой софт